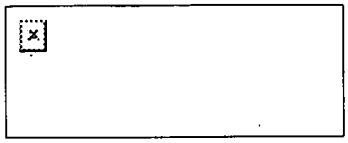


Special Edition Using HTML 4

← Previous Chapter → Next Chapter ↑ Contents

- 18 -
Positioning HTML Elements

by Jerry Honeycutt

Understanding CSS Positioning

In the desktop publishing world, layers are rectangular blocks of text and artwork that you can position anywhere on the page. You can also overlap layers so that one is hidden behind another or so that one bleeds through another. Publishers use layers to create some pretty awesome layouts. Take a look at any print advertisements or brochures, for example. Chances are, the publisher used layers.

While desktop publishers might take layering for granted (even the simplest of desktop publishing programs allow you to create and overlap layers), HTML designers don't. Because HTML is streaming, they've never had the capability to overlap blocks of text and artwork. That is, each preceding HTML element is displayed before the next--in order. HTML has never provided for the positioning of an HTML element, much less for overlapping HTML elements.

Until now--HTML 4.0 introduces the CSS positioning. If you're unsure of how to use style sheets, you should review Chapter 17, "Applying Cascading Style Sheets."

Positioning an HTML Element

You define an element's position using a style. Within a style's rule, you assign either `RELATIVE` or `ABSOLUTE` to the `POSITION` property. If you assign `RELATIVE`, the browser positions the element relative to its normal position (the location at which it would naturally appear). If you assign `ABSOLUTE` to `POSITION`, the browser positions the element relative to the parent container, whether it be the document or some other element, such as a `<DIV>` tag. In most cases, you use `ABSOLUTE` to place an element at an exact location relative to its parent. Use `RELATIVE` if you want to nudge an element slightly out of its normal location.

You position an element using a style's `TOP` and `LEFT` properties, like this:

```
#element {position: absolute; top: 100; left: 20}
```

When you assign the example style to an element, the browser positions it 100 pixels down and 20 pixels to the right of document's top, left-hand corner.

`LEFT` and `TOP` are represented in pixels, by default, and are relative to the top, left corner of the containing area within

the HTML document. For example, to position an element 10 pixels from the left edge of the browser window and 40 pixels from the top edge, use LEFT=10 and TOP=40. The browser draws the HTML document as though the positioned element does not exist, and then the element is overlapped with the Web page at the given offset. You can also assign percentages to LEFT and TOP, which represent a percentage of the parent container's width and height, respectively. Listing 18.1 positions an element in the middle of the Web page. As shown in Figure 18.1, the contents of the HTML document bleed through the contents of the positioned element because it's transparent.

Listing 18.1 Positioning an Element in the Middle of the Page

```
<HTML>
<HEAD>
<TITLE>Listing CC.1</TITLE>
</HEAD>
<STYLE TYPE="text/css">
  #example {position: absolute; top: 40; left: 100}
</STYLE>
<BODY>
<IMG ID=example SRC=init.gif>
<P>You can position an element anywhere you like.</P>
<P>This element is positioned, however, so that it overlaps
the HTML document below it. Notice how this text displays
through the image's transparent background.</P>
<P>This element is positioned, however, so that it overlaps
the HTML document below it. Notice how this text displays
through the image's transparent background.</P>
<P>This element is positioned, however, so that it overlaps
the HTML document below it. Notice how this text displays
through the image's transparent background.</P>
<P>This element is positioned, however, so that it overlaps
the HTML document below it. Notice how this text displays
through the image's transparent background.</P>
</BODY>
</HTML>
```

FIG. 18.1
You can position the element wherever you want on the browser window.

Changing the Size of an Element

You can change the height and width of the rectangular area occupied by an element. You use the WIDTH and HEIGHT properties. Like TOP and LEFT, you can assign a length or percentage to either of these properties. You can also assign AUTO to them and the browser automatically determines the appropriate width and height.

```
#element {position: absolute; top: 20; left: 20; width: 100; height: 100}
```

You don't use the WIDTH attribute to define the element's absolute width. This property suggests a width for purposes of wrapping the text contained within the element. If the text doesn't completely fill the element, however, the element is not actually as wide as the specified value. If you're inserting an image (or another element the browser can't wrap) inside an element, and the image is wider than the suggested width, the element's actual width is bigger than the suggested value.

TIP: The OVERFLOW property determines what the browser does in the case that an element's contents are

larger than the element's size. Assign NONE, and the contents are not clipped. Assign CLIP, and the browser chops off the content to fit in the rectangular area. Assign SCROLL, and the browser allows you to scroll the window so you can see more of it.

Listing 18.2 shows an example of an element that is 160 pixels wide and positioned 80 pixels from the top. As shown in Figure 18.2, the text wraps within the element, just like it would wrap within a table cell that's 160 pixels wide.

Listing 18.2 Specifying the Width of an Element

```
<HTML>
<HEAD>
<TITLE>Example 2</TITLE>
</HEAD>
<STYLE TYPE="text/css">
  #example {position: absolute; top: 80; left: 40; width: 160}
</STYLE>
<BODY>
<DIV ID=example>
  This text is within the DIV tag. Notice how its
  length is controlled by the width property.
</DIV>
</BODY>
</HTML>
```

TIP: You can use positioned elements to perform many of the same formatting tricks you've learned to do with the <TABLE> tag.

FIG. 18.2
You can leave either the TOP or LEFT attributes out, and the browser positions the element as though the omitted attribute is 0.

Overlapping Multiple Elements

You can cause elements to overlap by setting each element's TOP and LEFT attributes so that one appears on top of another. Figure 18.3 shows two elements. The first contains a handful of text and has a background image. The second element contains an image with a transparent background. The second element is positioned so that it overlaps the first (see Listing 18.3).

Listing 18.3 Overlapping Elements

```
<HTML>
<HEAD>
<TITLE>Example 3</TITLE>
</HEAD>
<STYLE TYPE="text/css">
  #example {position: absolute; top: 40; left: 60}
  #another {position: absolute; top: 80; left: 200}
</STYLE>
<BODY>
<DIV ID=example>
  <B>This is the first layer. It's behind the second layer.</B><BR>
```

```
  <B>This is the first layer. It's behind the second layer.</B><BR>
  <B>This is the first layer. It's behind the second layer.</B><BR>
  <B>This is the first layer. It's behind the second layer.</B><BR>
  <B>This is the first layer. It's behind the second layer.</B><BR>
  <B>This is the first layer. It's behind the second layer.</B><BR>
  <B>This is the first layer. It's behind the second layer.</B><BR>
  <B>This is the first layer. It's behind the second layer.</B><BR>
</DIV>
<IMG ID=another SRC=init.gif>
</BODY>
</HTML>
```

FIG. 18.3
Because the image in the second element has transparent areas, the content behind this element bleeds through.

> **NOTE:** By default, the browser draws overlapped elements in the order it encounters them. That is, it draws the first element, overlaps that with the next element, and so on.

If you don't like the order in which the browser overlaps elements, you can change it. The most straightforward way is by using the Z-INDEX property, which defines the stacking order for elements:

```
#example {position: absolute; top: 100; left: 100; z-index: 1}
```

You set this attribute to any positive integer value. An element with a stacking order larger than another draws over the other element. For example, an element with a stacking order of 10 overlaps an element with a stacking order of five. On the other hand, an element with a stacking order of three is overlapped by an element with a stacking order of five.

Listing 18.4 is an example of three elements, each of which uses the Z-INDEX attribute to define its stacking order. The first has a stacking order of two; the second has a stacking order of one, and the third has a stacking order of three. Thus, the browser draws the second element first, the first element second, and the third element last, as shown in Figure 18.4.

Listing 18.4 Using Z-INDEX

```
<HTML>
<HEAD>
<TITLE>Example 4</TITLE>
</HEAD>
<STYLE TYPE="text/css">
  #ex1 {position: absolute; top: 40; left: 60; z-index: 2}
  #ex2 {position: absolute; top: 30; left: 200; z-index: 1}
  #ex3 {position: absolute; top:100; left: 80; z-index: 3}
</STYLE>
<BODY>
<DIV ID=ex1>
<B>This is the first element. It's in the middle.</B><BR>
<B>This is the first element. It's in the middle.</B><BR>
<B>This is the first element. It's in the middle.</B><BR>
<B>This is the first element. It's in the middle.</B><BR>
<B>This is the first element. It's in the middle.</B><BR>
<B>This is the first element. It's in the middle.</B><BR>
<B>This is the first element. It's in the middle.</B><BR>
<B>This is the first element. It's in the middle.</B><BR>
```

```
</DIV>
<DIV ID=ex2>
<B>This is the second layer. It's behind the first layer.</B><BR>
<B>This is the second layer. It's behind the first layer.</B><BR>
<B>This is the second layer. It's behind the first layer.</B><BR>
<B>This is the second layer. It's behind the first layer.</B><BR>
<B>This is the second layer. It's behind the first layer.</B><BR>
<B>This is the second layer. It's behind the first layer.</B><BR>
<B>This is the second layer. It's behind the first layer.</B><BR>
<B>This is the second layer. It's behind the first layer.</B><BR>
</DIV>
<IMG ID=ex3 SRC=init.gif>
</LAYER>
</BODY>
</HTML>
```

FIG. 18.4
The `Z-INDEX` property essentially defines the order in which each element is drawn.

TIP: You can overlap several elements at the same position, define each element's stacking order in sequence, and then peel away the elements one at a time (using a script) to create a simple animation.

Nesting Positioned Elements

So far, you've only seen cases in which a handful of elements were added to the HTML document. They were siblings in that one was not contained within another. You can insert one element inside another, however, to create a parent-child relationship. In that case, the child (inside) is relative to the parent (outside). Thus, if you position an element called PARENT and locate it at 10, 10; and nest an element inside of PARENT called CHILD located at 5, 5; the child element actually displays at 15, 15 on the HTML document. If you move the parent element to 20, 20, the child element moves right along with it to 25, 25.

Listing 18.5 shows you an example of nested elements. The parent element contains an image of a rough Christmas tree. It contains a number of nested elements that represent bulbs. The coordinates of each nested element are relative to the upper-left corner of the parent. If you moved the Christmas tree to another location on the Web page, the bulbs would move right along with it (see Figure 18.5).

Listing 18.5 esting Elements

```
<HTML>
<HEAD>
<TITLE>Example 5</TITLE>
</HEAD>
<BODY>
<DIV STYLE="position: absolute; top: 100; left: 100">
<IMG SRC=xtree.gif>
  <IMG STYLE="position: absolute; top: 140; left: 60" SRC=ball1.gif>
  <IMG STYLE="position: absolute; top: 20; left: 100" SRC=ball2.gif>
  <IMG STYLE="position: absolute; top: 130; left: 120" SRC=ball1.gif>
  <IMG STYLE="position: absolute; top: 170; left: 140" SRC=ball2.gif>
  <IMG STYLE="position: absolute; top: 200; left: 120" SRC=ball2.gif>
  <IMG STYLE="position: absolute; top: 80; left: 80" SRC=ball3.gif>
  <IMG STYLE="position: absolute; top: 90; left: 125" SRC=ball3.gif>
  <IMG STYLE="position: absolute; top: 200; left: 60" SRC=ball3.gif>
```

```
  <IMG STYLE="position: absolute; top: 200; left: 180" SRC=ball3.gif>
</DIV>
</BODY>
</HTML>
```

FIG. 18.5
By capturing the mouse events for each bulb, you can allow the user to move the bulbs around on the Christmas tree.

Positioning Elements with Scripts

You get a lot of publishing capabilities just from being able to position an element anywhere on an HTML document and overlap it with others. You can create a variety of special effects, however, by attaching a script to an element and using that script to hide or show it--you can even move it around the browser window.

Given an element's ID, you reference it using the object model (see Chapter 19, "Scripting the Object Model"), like this:

```
document.all.item(id, 0)
```

id is the element's ID. The `0` indicates that you want the first occurrence of any element with that particular ID.

To access one of the styles, use the `style` object by appending `.style` to the object representing the element; then append the name of the style you're accessing to the end of that. Here's an example:

```
document.all.item(id, 0).style.property
```

Using a Script to Hide or Show an Element

You can use a script to hide and show elements on the HTML document. For example, you can create an element that you only want to display when the user moves the mouse across an image. In that case, you'd set the element's `VISIBILITY` property to `"hidden"` so that it's not initially displayed. Then, in the image's `OnMouseOver` event, set the element's `visibility` property to `""`, like this:

```
document.all.item(id, 0).style.property = "";
```

Listing 18.6 shows you an example that does something similar. It uses JavaScript, but you can use another scripting language just as well. It contains three elements and three buttons. The script associated with each button toggles the visibility of each element. Click a button associated with a visible element and the script makes the element invisible (see Figure 18.6).

Take a look at the function called `ToggleFirst()`. It toggles the state of the flag called `ShowFirst`, which indicates whether the element called `FIRST` is visible. It then sets the element's `visibility` property to `"hidden"` if `ShowFirst` is `false`; otherwise, it sets the property to `""`, which causes the element to become visible.

Listing 18.6 Hiding and Showing Elements

```
<HTML>
<HEAD>
<TITLE>Example 6</TITLE>
</HEAD>
<STYLE TYPE="text/css">
```

```
  #first {position: absolute; top: 80; left: 60}
  #second {position: absolute; top: 120; left: 200}
  #third {position: absolute; top: 140; left: 180}
</STYLE>
<SCRIPT LANGUAGE=JAVASCRIPT>
  ShowFirst = true;
  ShowSecond=false;
  ShowThird=true;
  function ToggleFirst()
  {
    ShowFirst = !ShowFirst;
    document.all.item("first", 0).style.visibility = ShowFirst ? "" :
        "hidden";
  }
  function ToggleSecond()
  {
    ShowSecond = !ShowSecond;
    document.all.item("second", 0).style.visibility = ShowSecond ? "" : "hidden";
  }
  function ToggleThird()
  {
    ShowThird = !ShowThird;
    document.all.item("third", 0).style.visibility = ShowThird ? "" :
        "hidden";
  }
</SCRIPT>
<BODY>
<DIV ID=first>
<B>This is the first layer. It's in the middle.</B><BR>
<B>This is the first layer. It's in the middle.</B><BR>
<B>This is the first layer. It's in the middle.</B><BR>
<B>This is the first layer. It's in the middle.</B><BR>
<B>This is the first layer. It's in the middle.</B><BR>
<B>This is the first layer. It's in the middle.</B><BR>
<B>This is the first layer. It's in the middle.</B><BR>
<B>This is the first layer. It's in the middle.</B><BR>
</DIV>
<DIV ID=second STYLE="visibility: hidden">
<B>This is the second layer. It's behind the first layer.</B><BR>
<B>This is the second layer. It's behind the first layer.</B><BR>
<B>This is the second layer. It's behind the first layer.</B><BR>
<B>This is the second layer. It's behind the first layer.</B><BR>
<B>This is the second layer. It's behind the first layer.</B><BR>
<B>This is the second layer. It's behind the first layer.</B><BR>
<B>This is the second layer. It's behind the first layer.</B><BR>
<B>This is the second layer. It's behind the first layer.</B><BR>
</DIV>
<IMG ID=third SRC=init.gif>
<FORM NAME=TOGGLE>
  <TABLE ALIGN=CENTER>
    <TD>
      <INPUT NAME=FIRST TYPE=BUTTON VALUE="Toggle First Layer "        onclick="ToggleFirst();">
    </TD>
    <TD>
      <INPUT NAME=SECOND TYPE=BUTTON VALUE="Toggle Second Layer"        onclick="ToggleSecond();
    </TD>
    <TD>
      <INPUT NAME=THIRD TYPE=BUTTON VALUE="Toggle Third Layer "        onclick="ToggleThird();">
    </TD>
  </TABLE>
```

```
</FORM>
</BODY>
</HTML>
```

FIG. 18.6
As you click buttons to hide an element, the browser peels that layer away, unveiling what's underneath.

TIP: In Windows, you've seen dialog boxes that contain a button with the text `More>>`. When you click that button, additional fields are presented. You can achieve the same effect in an HTML form by attaching a script to a form's button that shows another form hidden within a `<DIV>` tag.

Moving an Element with a Script

Besides showing and hiding an element, you can also move it around on the Web page. You can use this to create some pretty fancy animation, such as a curtain that appears to open, unveiling the contents of the page. Moving an element around is easy. You set the value of the `left` and `top` properties (using `posLeft` and `posTop` so that you don't have to change the length as a string with units), as shown in Listing 18.7 and Figure 18.7.

This example contains two elements. It also contains four buttons, label `Up`, `Down`, `Left`, and `Right`. Each button is associated with a function that moves the second element in the appropriate direction. For example, the `Up` function subtracts 10 from the second element's `top` property, which has the effect of moving the element up 10 pixels. The `Right` function adds 10 to the second element's `left` property, which has the effect of moving the element right 10 pixels.

Listing 18.7 Moving an Element with a Script

```
<HTML>
<HEAD>
<TITLE>Example 7</TITLE>
<SCRIPT LANGUAGE=JAVASCRIPT>
  function Up()
  {
    document.all.item("second", 0).style.posTop -= 10;
  }
  function Down()
  {
    document.all.item("second", 0).style.posTop += 10;
  }
  function Left()
  {
    document.all.item("second", 0).style.posLeft -= 10;
  }
  function Right()
  {
    document.all.item("second", 0).style.posLeft += 10;
  }
</SCRIPT>
</HEAD>
<BODY>
<DIV STYLE="position: absolute; top: 200; left: 300"; z-order: 2">
<B>This is the first layer. It's always on top.</B><BR>
<B>This is the first layer. It's always on top.</B><BR>
<B>This is the first layer. It's always on top.</B><BR>
```

```
<B>This is the first layer. It's always on top.</B><BR>
<B>This is the first layer. It's always on top.</B><BR>
<B>This is the first layer. It's always on top.</B><BR>
<B>This is the first layer. It's always on top.</B><BR>
<B>This is the first layer. It's always on top.</B><BR>
<B>This is the first layer. It's always on top.</B><BR>
</DIV>
<IMG ID=second STYLE="position: absolute; top: 180; left: 0; z-order: 1" SRC=init.gif>
<FORM NAME=BUTTONS>
<TABLE>
<TR>
<TD></TD>
<TD ALIGN=CENTER>
<INPUT WIDTH=100% NAME=UP TYPE=BUTTON VALUE="Up" onclick="Up();">
</TD>
<TD></TD>
</TR>
<TR>
<TD ALIGN=CENTER>
<INPUT NAME=LEFT TYPE=BUTTON VALUE="Left " onclick="Left();">
</TD>
<TD></TD>
<TD ALIGN=CENTER>
<INPUT WIDTH=100 NAME=RIGHT TYPE=BUTTON VALUE="Right" onclick="Right();">
</TD>
</TR>
<TR>
<TD></TD>
<TD ALIGN=CENTER>
<INPUT WIDTH=100 NAME=DOWN TYPE=BUTTON VALUE="Down " onclick="Down();">
</TD>
<TD></TD>
</TR>
</TABLE>
</FORM>
</BODY>
</HTML>
```

Expanding Forms: An Example

There are two types of users in this world: basic and advanced. With forms, you find many cases in which a basic user needs to fill in only a few of the simpler fields, while the advanced user needs to fill in all of the fields, including the more advanced fields.

You can display all of the form's fields at one time and let the basic user ignore the advanced fields or you can hide the advanced fields and let the advanced user get to them by clicking a special button. The latter is the approach that Windows 95 and Windows NT 4.0 take in many cases. Have you ever seen a dialog box in Windows with a button labeled `Advanced` or `More`? When the user clicks one of these buttons, the dialog box unfolds to show more fields.

With CSS positioning, hiding a portion of a form until a user clicks a button is easy. Listing 18.8 is just such an example. At the bottom of this HTML document, you see a portion of the form sandwiched in a `<DIV>` tag. The `<DIV>` tag's `visibility` property is initially set to `"hidden"`. The function `OpenMore` toggles the visibility of the hidden portion of the form and changes the text displayed in the button to reflect the state of the form.

FIG. 18.7

As you move the second element by the first, it disappears under the first element, because the second element has a larger `z-order`.

Listing 18.8 Expanding a Form

```
<HTML>
<HEAD>
<TITLE>Example 8</TITLE>
<SCRIPT LANGUAGE=JAVASCRIPT>
blnMoreIsUp = false;
// Display the hidden form if it's not already displayed. Otherwise, hide it.
// Also, change the text in the button to reflect the current state of the hidden form.
function OpenMore()
{
  blnMoreIsUp = !blnMoreIsUp;
  document.all.item("MORE", 0).style.visibility = blnMoreIsUp ? "" : "hidden";
  document.FEEDBACK.FEEDBACK_MORE.value = blnMoreIsUp ? "Less<<" : "More>>";
}
</SCRIPT>
</HEAD>
<BODY>
<FORM NAME=FEEDBACK METHOD=GET ACTION="mailto:jerry@honeycutt.com" OnSubmit="return IsValid()">
  <TABLE CELLPADDING=10>
   <TR>
      <TD VALIGN=TOP>
         <B>Please provide your e-mail address:</B><BR>
         <INPUT NAME=FEEDBACK_MAIL TYPE=TEXT SIZE=40>
      </TD>
      <TD VALIGN=TOP>
        <B>How did you find our site:</B><BR>
        <SELECT NAME=FEEDBACK_HOW SIZE=1>
           <OPTION VALUE=1>AltaVista
           <OPTION VALUE=2>Excite
           <OPTION VALUE=3>Lycos
           <OPTION VALUE=4>Yahoo!
           <OPTION VALUE=5>WebCrawler
           <OPTION VALUE=6>Friend
           <OPTION VALUE=7>Other Link
      </SELECT>
      </TD>
    <TR>
      <TD VALIGN=TOP ROWSPAN=2>
        <B>Tell us what you think about our Web site:</B><BR>
        <TEXTAREA NAME=FEEDBACK_MEMO COLS=45 ROWS=8>
        </TEXTAREA>
      </TD>
      <TD VALIGN=TOP>
        <B>How did we rate?</B><BR>
        <TABLE BORDER=1>
          <TR ALIGN=CENTER>
            <TH></TH><TH>Yes</TH><TH>No</TH>
          </TR>
          <TR ALIGN=CENTER>
            <TD ALIGN=LEFT>
              Did this site load fast enough?
            </TD>
            <TD>
              <INPUT NAME=FEEDBACK_SPEED TYPE=RADIO>
            </TD>
```

```
            <TD>
              <INPUT NAME=FEEDBACK_SPEED TYPE=RADIO>
            </TD>
          </TR>
          <TR ALIGN=CENTER>
            <TD ALIGN=LEFT>
              Did you find the graphics interesting?
            </TD>
            <TD>
              <INPUT NAME=FEEDBACK_GRAPHIC TYPE=RADIO>
            </TD>
            <TD>
              <INPUT NAME=FEEDBACK_GRAPHIC TYPE=RADIO>
            </TD>
          </TR>
          <TR ALIGN=CENTER>
            <TD ALIGN=LEFT>
                Was the content suitable?
            </TD>
            <TD>
              <INPUT NAME=FEEDBACK_CONTENT TYPE=RADIO>
            </TD>
            <TD>
              <INPUT NAME=FEEDBACK_CONTENT TYPE=RADIO>
            </TD>
          </TR>
        </TABLE>
      </TD>
    </TR>
    <TR ALIGN=RIGHT>
      <TD>
        <TABLE WIDTH=100%>
          <TD ALIGN=LEFT>
            <INPUT NAME=FEEDBACK_MORE TYPE=BUTTON VALUE="More>>"                OnClick="OpenMor
          </TD>
          <TD>
            <INPUT NAME=FEEDBACK_RESET TYPE=RESET VALUE=Clear>
            <INPUT NAME=FEEDBACK_SUBMIT TYPE=SUBMIT VALUE=Submit>
          </TD>
        </TABLE>
      </TD>
    </TR>
  </TABLE>
  <!-- This DIV contains the hidden part of the form that the user sees when          they click
  <DIV ID=MORE STYLE="visibility: hidden">
    <TABLE CELLPADDING=10>
      <TR>
        <TD>
          <B>Type the URL of your home page:</B><BR>
          <INPUT NAME=FEEDBACK_URL TYPE=TEXT SIZE=60>
        </TD>
        <TD>
          <B>Type your phone number:</B><BR>
          <INPUT NAME=FEEDBACK_PHONE TYPE=TEXT SIZE=32>
        </TD>
      </TR>
    </TABLE>
  </DIV>
</FORM>
</BODY>
```

```
</HTML>
```

Figure 18.8 shows the resulting form with the hidden form expanded.

FIG. 18.8
If the user clicks `Less<<` to hide the advanced fields, the browser still keeps any data that the user typed in these fields.

Previous Chapter | Next Chapter | Contents

INTERNET ARCHIVE
Wayback Machine

Enter Web Address: http:// All Take Me Back Adv. Search Compare Archive Pages

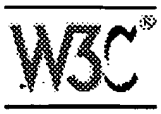
Searched for http://docs.rinet.ru/HTML4/ch18/ch18.htm **11** Results

* denotes when site was updated.

Search Results for Jan 01, 1996 - Feb 09, 2005

1996	1997	1998	1999	2000	2001	2002	2003	2004	2005
0 pages	0 pages	0 pages	1 pages	4 pages	3 pages	0 pages	2 pages	1 pages	0 pages
			Nov 14, 1999 *	Mar 11, 2000	Mar 04, 2001 *		Dec 08, 2003 *	Feb 25, 2004 *	
				Jun 17, 2000	Apr 14, 2001		Dec 24, 2003		
				Oct 05, 2000	Jul 25, 2001 *				
				Dec 06, 2000					

Home | Help



HTML 4.01 Specification

W3C Recommendation 24 December 1999

This version:
http://www.w3.org/TR/1999/REC-html401-19991224
(plain text [794Kb], gzip'ed tar archive of HTML files [371Kb], a .zip archive of HTML files [405Kb], gzip'ed Postscript file [746Kb, 389 pages], gzip'ed PDF file [963Kb])
Latest version of HTML 4.01:
http://www.w3.org/TR/html401
Latest version of HTML 4:
http://www.w3.org/TR/html4
Latest version of HTML:
http://www.w3.org/TR/html
Previous version of HTML 4.01:
http://www.w3.org/TR/1999/PR-html40-19990824
Previous HTML 4 Recommendation:
http://www.w3.org/TR/1998/REC-html40-19980424
Editors:
Dave Raggett <dsr@w3.org>
Arnaud Le Hors, W3C
Ian Jacobs, W3C

Abstract

This specification defines the HyperText Markup Language (HTML), the publishing language of the World Wide Web. This specification defines HTML 4.01, which is a subversion of HTML 4. In addition to the text, multimedia, and hyperlink features of the previous versions of HTML (HTML 3.2 [HTML32] [p.356] and HTML 2.0 [RFC1866] [p.356]), HTML 4 supports more multimedia options, scripting languages, style sheets, better printing facilities, and documents that are more accessible to users with disabilities. HTML 4 also takes great strides towards the internationalization of documents, with the goal of making the Web truly World Wide.

HTML 4 is an SGML application conforming to International Standard ISO 8879 -- Standard Generalized Markup Language [ISO8879] [p.353] .

16 Frames

Contents

16.1 Introduction to frames

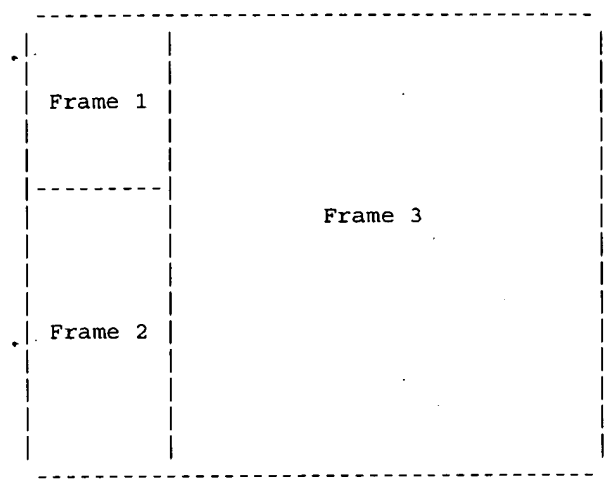
HTML frames allow authors to present documents in multiple views, which may be independent windows or subwindows. Multiple views offer designers a way to keep certain information visible, while other views are scrolled or replaced. For example, within the same window, one frame might display a static banner, a second a navigation menu, and a third the main document that can be scrolled through or replaced by navigating in the second frame.

Here is a simple frame document:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
   "http://www.w3.org/TR/html4/frameset.dtd">
<HTML>
<HEAD>
<TITLE>A simple frameset document</TITLE>
</HEAD>
<FRAMESET cols="20%, 80%">
  <FRAMESET rows="100, 200">
      <FRAME src="contents_of_frame1.html">
      <FRAME src="contents_of_frame2.gif">
  </FRAMESET>
  <FRAME src="contents_of_frame3.html">
  <NOFRAMES>
      <P>This frameset document contains:
      <UL>
         <LI><A href="contents_of_frame1.html">Some neat contents</A>
         <LI><IMG src="contents_of_frame2.gif" alt="A neat image">
```

```
        <LI><A href="contents_of_frame3.html">Some other neat contents</A>
      </UL>
  </NOFRAMES>
</FRAMESET>
</HTML>
```

that might create a frame layout something like this:



If the user agent can't display frames or is configured not to, it will render the contents of the NOFRAMES element.

16.2 Layout of frames

An HTML document that describes frame layout (called a *frameset document*) has a different makeup than an HTML document without frames. A standard document has one HEAD section and one BODY. A frameset document has a HEAD, and a FRAMESET in place of the BODY.

The FRAMESET section of a document specifies the layout of views in the main user agent window. In addition, the FRAMESET section can contain a NOFRAMES element to provide alternate content [p.214] for user agents that do not support frames or are configured not to display frames.

Elements that might normally be placed in the BODY element must not appear before the first FRAMESET element or the FRAMESET will be ignored.

16.2.1 The FRAMESET element

```
<![ %HTML.Frameset; [
<!ELEMENT FRAMESET - - ((FRAMESET|FRAME)+ & NOFRAMES?) -- window subdivision-->
<!ATTLIST FRAMESET
  %coreattrs;                          -- id, class, style, title --
  rows        %MultiLengths; #IMPLIED  -- list of lengths,
                                          default: 100% (1 row) --
  cols        %MultiLengths; #IMPLIED  -- list of lengths,
                                          default: 100% (1 col) --
```

```
  onload      %Script;       #IMPLIED  -- all the frames have been loaded  --
  onunload    %Script;       #IMPLIED  -- all the frames have been removed --
  >
]]>
```

Attribute definitions

`rows` = *multi-length-list* [p.52] [CN] [p.49]
: This attribute specifies the layout of horizontal frames. It is a comma-separated list of pixels, percentages, and relative lengths. The default value is 100%, meaning one row.

`cols` = *multi-length-list* [p.52] [CN] [p.49]
: This attribute specifies the layout of vertical frames. It is a comma-separated list of pixels, percentages, and relative lengths. The default value is 100%, meaning one column.

Attributes defined elsewhere

- `id`, `class` (document-wide identifiers [p.71])
- `title` (element title [p.63])
- `style` (inline style information [p.186])
- `onload`, `onunload` (intrinsic events [p.254])

The `FRAMESET` element specifies the layout of the main user window in terms of rectangular subspaces.

Rows and columns

Setting the `rows` attribute defines the number of horizontal subspaces in a frameset. Setting the `cols` attribute defines the number of vertical subspaces. Both attributes may be set simultaneously to create a grid.

If the `rows` attribute is not set, each column extends the entire length of the page. If the `cols` attribute is not set, each row extends the entire width of the page. If neither attribute is set, the frame takes up exactly the size of the page.

Frames are created left-to-right for columns and top-to-bottom for rows. When both attributes are specified, views are created left-to-right in the top row, left-to-right in the second row, etc.

The first example divides the screen vertically in two (i.e., creates a top half and a bottom half).

```
<FRAMESET rows="50%, 50%">
...the rest of the definition...
</FRAMESET>
```

The next example creates three columns: the second has a fixed width of 250 pixels (useful, for example, to hold an image with a known size). The first receives 25% of the remaining space and the third 75% of the remaining space.

```
<FRAMESET cols="1*,250,3*">
...the rest of the definition...
</FRAMESET>
```

The next example creates a 2x3 grid of subspaces.

```
<FRAMESET rows="30%,70%" cols="33%,34%,33%">
...the rest of the definition...
</FRAMESET>
```

For the next example, suppose the browser window is currently 1000 pixels high. The first view is allotted 30% of the total height (300 pixels). The second view is specified to be exactly 400 pixels high. This leaves 300 pixels to be divided between the other two frames. The fourth frame's height is specified as "2*", so it is twice as high as the third frame, whose height is only "*" (equivalent to 1*). Therefore the third frame will be 100 pixels high and the fourth will be 200 pixels high.

```
<FRAMESET rows="30%,400,*,2*">
...the rest of the definition...
</FRAMESET>
```

Absolute lengths that do not sum to 100% of the real available space should be adjusted by the user agent. When underspecified, remaining space should be allotted proportionally to each view. When overspecified, each view should be reduced according to its specified proportion of the total space.

Nested frame sets

Framesets may be nested to any level.

In the following example, the outer FRAMESET divides the available space into three equal columns. The inner FRAMESET then divides the second area into two rows of unequal height.

```
<FRAMESET cols="33%, 33%, 34%">
    ...contents of first frame...
    <FRAMESET rows="40%, 50%">
        ...contents of second frame, first row...
        ...contents of second frame, second row...
    </FRAMESET>
    ...contents of third frame...
</FRAMESET>
```

Sharing data among frames

Authors may share data among several frames by including this data via an OBJECT element. Authors should include the OBJECT element in the HEAD element of a frameset document and name it with the id attribute. Any document that is the contents of a frame in the frameset may refer to this identifier.

The following example illustrates how a script might refer to an OBJECT element defined for an entire frameset:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
   "http://www.w3.org/TR/html4/frameset.dtd">
<HTML>
<HEAD>
<TITLE>This is a frameset with OBJECT in the HEAD</TITLE>
<!-- This OBJECT is not rendered! -->
<OBJECT id="myobject" data="data.bar"></OBJECT>
</HEAD>
<FRAMESET>
    <FRAME src="bianca.html" name="bianca">
</FRAMESET>
</HTML>

<!-- In bianca.html -->
<HTML>
<HEAD>
<TITLE>Bianca's page</TITLE>
</HEAD>
<BODY>
...the beginning of the document...
<P>
<SCRIPT type="text/javascript">
parent.myobject.myproperty
</SCRIPT>
...the rest of the document...
</BODY>
</HTML>
```

16.2.2 The FRAME element

```
<![ %HTML.Frameset; [
<!-- reserved frame names start with "_" otherwise starts with letter -->
<!ELEMENT FRAME - O EMPTY                -- subwindow -->
<!ATTLIST FRAME
  %coreattrs;                            -- id, class, style, title --
  longdesc    %URI;          #IMPLIED    -- link to long description
                                            (complements title) --
  name        CDATA          #IMPLIED    -- name of frame for targetting --
  src         %URI;          #IMPLIED    -- source of frame content --
  frameborder (1|0)          1           -- request frame borders? --
  marginwidth %Pixels;       #IMPLIED    -- margin widths in pixels --
  marginheight %Pixels;      #IMPLIED    -- margin height in pixels --
  noresize    (noresize)     #IMPLIED    -- allow users to resize frames? --
  scrolling   (yes|no|auto)  auto        -- scrollbar or none --
  >
]]>
```

Attribute definitions

`name` = *cdata* [p.50] [CI] [p.49]
: This attribute assigns a name to the current frame. This name may be used as the target of subsequent links.

`longdesc` = *uri* [p.51] [CT] [p.49]
: This attribute specifies a link to a long description of the frame. This description should supplement the short description provided using the `title` attribute, and

may be particularly useful for non-visual user agents.

`src` = *uri* [p.51] [CT] [p.49]
: This attribute specifies the location of the initial contents to be contained in the frame.

`noresize` [CI] [p.49]
: When present, this boolean attribute tells the user agent that the frame window must not be resizeable.

`scrolling` = `auto` | `yes` | `no` [CI] [p.49]
: This attribute specifies scroll information for the frame window. Possible values
  - `auto`: This value tells the user agent to provide scrolling devices for the frame window when necessary. This is the default value.
  - `yes`: This value tells the user agent to always provide scrolling devices for the frame window.
  - `no`: This value tells the user agent not to provide scrolling devices for the frame window.

`frameborder` = `1` | `0` [CN] [p.49]
: This attribute provides the user agent with information about the frame border. Possible values:
  - `1`: This value tells the user agent to draw a separator between this frame and every adjoining frame. This is the default value.
  - `0`: This value tells the user agent not to draw a separator between this frame and every adjoining frame. Note that separators may be drawn next to this frame nonetheless if specified by other frames.

`marginwidth` = *pixels* [p.52] [CN] [p.49]
: This attribute specifies the amount of space to be left between the frame's contents in its left and right margins. The value must be greater than zero (pixels). The default value depends on the user agent.

`marginheight` = *pixels* [p.52] [CN] [p.49]
: This attribute specifies the amount of space to be left between the frame's contents in its top and bottom margins. The value must be greater than zero (pixels). The default value depends on the user agent.

Attributes defined elsewhere

- `id`, `class` (document-wide identifiers [p.71])
- `title` (element title [p.63])
- `style` (inline style information [p.186])

The `FRAME` element defines the contents and appearance of a single frame.

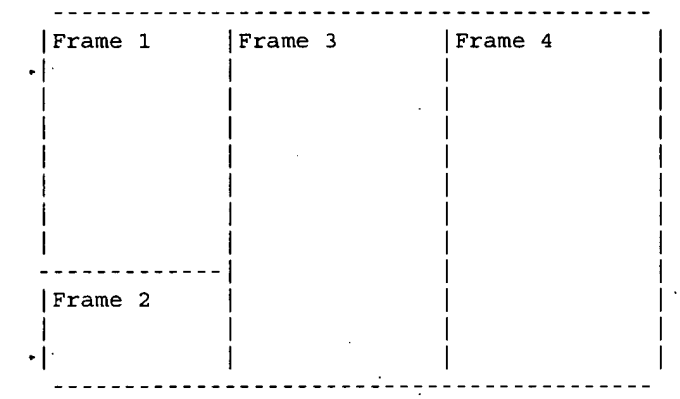
Setting the initial contents of a frame

The `src` attribute specifies the initial document the frame will contain.

The following example HTML document:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
   "http://www.w3.org/TR/html4/frameset.dtd">
<HTML>
<HEAD>
<TITLE>A frameset document</TITLE>
</HEAD>
<FRAMESET cols="33%,33%,33%">
  <FRAMESET rows="*,200">
      <FRAME src="contents_of_frame1.html">
      <FRAME src="contents_of_frame2.gif">
  </FRAMESET>
  <FRAME src="contents_of_frame3.html">
  <FRAME src="contents_of_frame4.html">
</FRAMESET>
</HTML>
```

should create a frame layout something like this:



and cause the user agent to load each file into a separate view.

The contents of a frame must not be in the same document as the frame's definition.

ILLEGAL EXAMPLE:
The following frameset definition is not legal HTML since the contents of the second frame are in the same document as the frameset.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
   "http://www.w3.org/TR/html4/frameset.dtd">
<HTML>
<HEAD>
<TITLE>A frameset document</TITLE>
</HEAD>
<FRAMESET cols="50%,50%">
  <FRAME src="contents_of_frame1.html">
  <FRAME src="#anchor_in_same_document">
  <NOFRAMES>
  ...some text...
  <H2><A name="anchor_in_same_document">Important section</A></H2>
```

```
  ...some text...
  </NOFRAMES>
</FRAMESET>
</HTML>
```

Visual rendering of a frame

The following example illustrates the usage of the decorative FRAME attributes. We specify that frame 1 will allow no scroll bars. Frame 2 will leave white space around its contents (initially, an image file) and the frame will not be resizeable. No border will be drawn between frames 3 and 4. Borders will be drawn (by default) between frames 1, 2, and 3.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
   "http://www.w3.org/TR/html4/frameset.dtd">
<HTML>
<HEAD>
<TITLE>A frameset document</TITLE>
</HEAD>
<FRAMESET cols="33%,33%,33%">
  <FRAMESET rows="*,200">
      <FRAME src="contents_of_frame1.html" scrolling="no">
      <FRAME src="contents_of_frame2.gif"
                marginwidth="10" marginheight="15"
                noresize>
  </FRAMESET>
  <FRAME src="contents_of_frame3.html" frameborder="0">
  <FRAME src="contents_of_frame4.html" frameborder="0">
</FRAMESET>
</HTML>
```

16.3 Specifying target frame information

Note. *For information about current practice in determining the target of a frame, please consult the notes on frames [p.350] in the appendix.*

Attribute definitions

`target` = *frame-target* [p.57] [CI] [p.49]
This attribute specifies the name of a frame where a document is to be opened.

By assigning a name to a frame via the `name` attribute, authors can refer to it as the "target" of links defined by other elements. The `target` attribute may be set for elements that create links (A, LINK), image maps (AREA), and forms (FORM).

Please consult the section on target frame names [p.57] for information about recognized frame names.

This example illustrates how targets allow the dynamic modification of a frame's contents. First we define a frameset in the document `frameset.html`, shown here:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
   "http://www.w3.org/TR/html4/frameset.dtd">
<HTML>
<HEAD>
<TITLE>A frameset document</TITLE>
</HEAD>
<FRAMESET rows="50%,50%">
   <FRAME name="fixed" src="init_fixed.html">
   <FRAME name="dynamic" src="init_dynamic.html">
</FRAMESET>
</HTML>
```

Then, in `init_dynamic.html`, we link to the frame named "dynamic".

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
<HTML>
<HEAD>
<TITLE>A document with anchors with specific targets</TITLE>
</HEAD>
<BODY>
...beginning of the document...
<P>Now you may advance to
    <A href="slide2.html" target="dynamic">slide 2.</A>
...more document...
<P>You're doing great. Now on to
    <A href="slide3.html" target="dynamic">slide 3.</A>
</BODY>
</HTML>
```

Activating either link opens a new document in the frame named "dynamic" while the other frame, "fixed", maintains its initial contents.

Note. *A frameset definition never changes, but the contents of one of its frames can. Once the initial contents of a frame change, the frameset definition no longer reflects the current state of its frames.*

There is currently no way to encode the entire state of a frameset in a URI. Therefore, many user agents do not allow users to assign a bookmark to a frameset.

Framesets may make navigation forward and backward through your user agent's history more difficult for users.

16.3.1 Setting the default target for links

When many links in the same document designate the same target, it is possible to specify the target once and dispense with the `target` attribute of each element. This is done by setting the `target` attribute of the `BASE` element.

We return to the previous example, this time factoring the target information by defining it in the `BASE` element and removing it from the `A` elements.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
<HTML>
<HEAD>
```

```
<TITLE>A document with BASE with a specific target</TITLE>
<BASE href="http://www.mycom.com/Slides" target="dynamic">
</HEAD>
<BODY>
...beginning of the document...
<P>Now you may advance to <A href="slide2.html">slide 2.</A>
...more document...
<P>You're doing great. Now on to
       <A href="slide3.html">slide 3.</A>
</BODY>
</HTML>
```

16.3.2 Target semantics

User agents should determine the target frame in which to load a linked resource according to the following precedences (highest priority to lowest):

1. If an element has its `target` attribute set to a known frame, when the element is activated (i.e., a link is followed or a form is processed), the resource designated by the element should be loaded into the target frame.
2. If an element does not have the `target` attribute set but the `BASE` element does, the `BASE` element's `target` attribute determines the frame.
3. If neither the element nor the `BASE` element refers to a target, the resource designated by the element should be loaded into the frame containing the element.
4. If any `target` attribute refers to an unknown frame `F`, the user agent should create a new window and frame, assign the name `F` to the frame, and load the resource designated by the element in the new frame.

User agents may provide users with a mechanism to override the `target` attribute.

16.4 Alternate content

Authors should supply alternate content for those user agents that do not support frames or are configured not to display frames.

16.4.1 The NOFRAMES element

```
<![ %HTML.Frameset; [
<!ENTITY % noframes.content "(BODY) -(NOFRAMES)">
]]>

<!ENTITY % noframes.content "(%flow;)*">

<!ELEMENT NOFRAMES - - %noframes.content;
 -- alternate content container for non frame-based rendering -->
<!ATTLIST NOFRAMES
  %attrs;                                -- %coreattrs, %i18n, %events --
  >
```

Attributes defined elsewhere

- `id`, `class` (document-wide identifiers [p.71])
- `lang` (language information [p.79]), `dir` (text direction [p.82])
- `title` (element title [p.63])
- `style` (inline style information [p.186])
- `onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, `onkeyup` (intrinsic events [p.254])

The NOFRAMES element specifies content that should be displayed only by user agents that do not support frames or are configured not to display frames. User agents that support frames must only display the contents of a NOFRAMES declaration when configured not to display frames. User agents that do not support frames must display the contents of NOFRAMES in any case.

The NOFRAMES element is part of both the transitional and frameset DTDs. [p.60] In a document that uses the frameset DTD, NOFRAMES may be used at the end of the FRAMESET section of the document.

For example:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
     "http://www.w3.org/TR/html4/frameset.dtd">
  <HTML>
  <HEAD>
  <TITLE>A frameset document with NOFRAMES</TITLE>
  </HEAD>
  <FRAMESET cols="50%, 50%">
     <FRAME src="main.html">
     <FRAME src="table_of_contents.html">
     <NOFRAMES>
     <P>Here is the <A href="main-noframes.html">
              non-frame based version of the document.</A>
     </NOFRAMES>
  </FRAMESET>
  </HTML>
```

NOFRAMES may be used, for example, in a document that is the source of a frame and that uses the transitional DTD. This allows authors to explain the document's purpose in cases when it is viewed out of the frameset or with a user agent that doesn't support frames.

16.4.2 Long descriptions of frames

The `longdesc` attribute allows authors to make frame documents more accessible to people using non-visual user agents. This attribute designates a resource that provides a long description of the frame. Authors should note that long descriptions associated with frames are attached to the *frame*, not the frame's contents. Since the contents may vary over time, the initial long description is likely to become inappropriate for the frame's later contents. In particular, authors should not include

an image as the sole content of a frame.

The following frameset document describes two frames. The left frame contains a table of contents and the right frame initially contains an image of an ostrich:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
   "http://www.w3.org/TR/html4/frameset.dtd">
<HTML>
<HEAD>
<TITLE>A poorly-designed frameset document</TITLE>
</HEAD>
<FRAMESET cols="20%, 80%">
   <FRAME src="table_of_contents.html">
   <FRAME src="ostrich.gif" longdesc="ostrich-desc.html">
</FRAMESET>
</HTML>
```

Note that the image has been included in the frame independently of any HTML element, so the author has no means of specifying alternate text other than via the `longdesc` attribute. If the contents of the right frame change (e.g., the user selects a rattlesnake from the table of contents), users will have no textual access to the frame's new content.

Thus, authors should not put an image directly in a frame. Instead, the image should be specified in a separate HTML document, and therein annotated with the appropriate alternate text:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
   "http://www.w3.org/TR/html4/frameset.dtd">
<HTML>
<HEAD>
<TITLE>A well-designed frameset document</TITLE>
</HEAD>
<FRAMESET cols="20%, 80%">
   <FRAME src="table_of_contents.html">
   <FRAME src="ostrich-container.html">
</FRAMESET>
</HTML>

<!-- In ostrich-container.html: -->
<HTML>
<HEAD>
<TITLE>The fast and powerful ostrich</TITLE>
</HEAD>
<P>
<OBJECT data="ostrich.gif" type="image/gif">
These ostriches sure taste good!
</OBJECT>
</HTML>
```

16.5 Inline frames: the IFRAME element

```
<!ELEMENT IFRAME - - (%flow;)*          -- inline subwindow -->
<!ATTLIST IFRAME
  %coreattrs;                           -- id, class, style, title --
  longdesc    %URI;          #IMPLIED   -- link to long description
                                           (complements title) --
  name        CDATA          #IMPLIED   -- name of frame for targetting --
  src         %URI;          #IMPLIED   -- source of frame content --
  frameborder (1|0)          1          -- request frame borders? --
  marginwidth %Pixels;       #IMPLIED   -- margin widths in pixels --
  marginheight %Pixels;      #IMPLIED   -- margin height in pixels --
  scrolling   (yes|no|auto)  auto       -- scrollbar or none --
  align       %IAlign;       #IMPLIED   -- vertical or horizontal alignment --
  height      %Length;       #IMPLIED   -- frame height --
  width       %Length;       #IMPLIED   -- frame width --
  >
```

Attribute definitions

`longdesc` = *uri* [p.51] [CT] [p.49]
: This attribute specifies a link to a long description of the frame. This description should supplement the short description provided using the `title` attribute, and is particularly useful for non-visual user agents.

`name` = *cdata* [p.50] [CI] [p.49]
: This attribute assigns a name to the current frame. This name may be used as the target of subsequent links.

`width` = *length* [p.52] [CN] [p.49]
: The width of the inline frame.

`height` = *length* [p.52] [CN] [p.49]
: The height of the inline frame.

Attributes defined elsewhere

- `id`, `class` (document-wide identifiers [p.71])
- `title` (element title [p.63])
- `style` (inline style information [p.186])
- `name`, `src`, `frameborder`, `marginwidth`, `marginheight`, `scrolling` (frame controls and decoration [p.209])
- `align` (alignment [p.195])

The `IFRAME` element allows authors to insert a frame within a block of text. Inserting an inline frame within a section of text is much like inserting an object via the `OBJECT` element: they both allow you to insert an HTML document in the middle of another, they may both be aligned with surrounding text, etc.

The information to be inserted inline is designated by the `src` attribute of this element. The *contents* of the `IFRAME` element, on the other hand, should only be displayed by user agents that do not support frames or are configured not to display frames.

For user agents that support frames, the following example will place an inline frame surrounded by a border in the middle of the text.

```
<IFRAME src="foo.html" width="400" height="500"
          scrolling="auto" frameborder="1">
[Your user agent does not support frames or is currently configured
not to display frames. However, you may visit
<A href="foo.html">the related document.</A>]
</IFRAME>
```

Inline frames may not be resized (and thus, they do not take the `noresize` attribute).

Note. *HTML documents may also be embedded in other HTML documents with the `OBJECT` element. See the section on embedded documents [p.173] for details.*